VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practise in the Company

2009 Petr Kubát

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2009 .....

Abstrakt

Práce se týká prozkoumání několika technologií pro společnost Kvados v rámci individuální odborné praxe. Prvním tématem je RIA framework Silverlight verze 2. Volby a schopnosti DataBindingu, spojení s webovými službami pro přístup k databázi, možnosti validace vstupních dat formulářových prvků Silverlightu. Další technoligií je Windows Workflow Foundation, její základní funkce a možnosti. Obě zmíněné technologie pochází od společnosti Microsoft.

Klíčová slova: praxe, csharp, xaml, microsoft, silverlight, workflow foundation

Abstract

Thesis is concerned with examination of several technologies for Kvados company within an individual professional practise. First topic is RIA framework Silverlight 2. Options and features of DataBinding, connecting to web services to access the database, possibilities of data validation of input controls in Silverlight. Next examined technology is Windows Workflow Foundation, its basic functions and features. Both mentioned technologies are made by Microsoft company.

Keywords: practise, csharp, xaml, microsoft, silverlight, workflow foundation

Seznam použitých zkratek a symbolů

CRM	–	Customer Relationship Management
DOM	–	Document Object Model
ERP	–	Enterprise Resource Planning
HTTP	–	Hypertext Transfer Protocol
RC	–	Release Candidate
RIA	–	Rich Internet Application
RTW	–	Release To Web
SL	–	Silverlight
SOAP	–	Simple Object Access Protocol
VB	–	Visual Basic
VS2008	–	Visual Studio 2008
WCF	–	Windows Communication Foundation
WF	–	Windows Workflow Foundation
WPF	–	Windows Presentation Foundation
WS	–	Webová služba
XAML	–	eXtensibile Application Markup Language

Obsah

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Má odborná praxe proběhla ve společnosti Kvados, a.s. Během ní jsem byl postaven před několik úkolů. Převážně se týkaly studia nových technologií, zkoumání jejich možností, nalezení hranic a důležitých problémů těchto technologií.

Nejdéle jsem se věnoval technologi Silverlight od firmy Microsoft. Tato technologie byla při mém nástupu právě ve stádiu vývoje nové přepracované verze 2. Během mého studia se Silverlight stále vyvíjel, upravoval, byly vydávány nové knihovny. Na konci praxe byla dokonce uvolněna třetí verze se spoustou nových prvků a přístupů, které usnadňují tvorbu RIA aplikací.

V části letního semestru jsem se zaměřil na technologii Windows Workflow Foundation, také od Microsoftu. WF je o něco starší technologie oproti Silverlightu, nicméně stále se jedná o technologi vydanou na konci roku 2006 jako součást .NET Frameworku verze 3.0.

1.1 Zaměření firmy Kvados, a.s.

> „Klíčovým produktem společnosti Kvados, a.s. je od roku 1998 produkt VENTUS. Jedná se o komplexní ERP řešení, které dnes pokrývá všechny běžné finanční, logistické i distribuční procesy. Obsahuje specializovaná řešení pro celní agendy, manažerské informační systémy, CRM a řešení pro obchodní zástupce v terénu.
>
> Společnost dále produkuje mobilní informační systém myAVIS^{TM}. Od uvedení tohoto produktu na trh patří společnost k leaderům trhu nejen v České republice, ale i Evropě. Řešení myAVIS^{TM} pokrývá všechny běžné procesy práce mobilních pracovníků v terénu a řeší oboustrannou replikaci dat s centrálním systémem, jejich vyhodnocení a následnou integraci do provozovaných řešení typu CRM a ERP.“ [1]

1.2 Pracovní zařazení

Jako praktikant jsem byl zařazen do nevývojové části společnosti. Nepodílel jsem se tedy na žádném firemním projektu nebo jeho součásti. Mým úkolem v rámci firmy bylo prozkoumat možnosti některých technologií, které by eventuálně mohla firma v budoucnu využít.

1.3 Zadané úkoly

V průběhu 2 semestrů jsem měl za úkol prozkoumat následující body:

1. Technologie Silverlight:
   - možnosti DataBindingu v Silverlightu ve spojení různými prvky,

- způsoby komunikace s webovými službami a navrhnout řešení k případným problémům,
- spolupráce Silverlight klienta s databází a provázání zobrazovaných dat a dat z databáze,
- varianty validace vstupu ve formulářích.

2. Technologie Windows Workflow Foundation:
   - komunikace mezi workflow a hostující aplikací,
   - zprovoznit ukázkové řešení implementace persistent service,
   - ukázkové řešení implementace tracking service,
   - spojení persistnece service a tracking service.

2 Řešení úkolů

2.1 Silverlight

Silverlight verze 1 byl založen na JavaScriptu spolu s XAML. Samotný JavaScript nebyl příliš vhodný a navíc SL1 postrádal jakékoliv vlastní ovládací prvky. Standartní prvky byly vykreslovány za pomoci DOM stránky HTML. Začal tedy vývoj Silverlight 1.1. Přechod na C# (VB) byl ale tak významný, že bylo číslo verze změněno na 2.

Silverlight ve druhé verzi vychází z technologie WPF - tedy C# + XAML za pomoci (části) .NET Frameworku. XAML slouží pro definici vzhledu aplikace, C# pro logiku aplikace (označuje se jako „kód v pozadí“). [2]

2.1.1 DataBinding

Mým prvním úkolem bylo vytvořit jednoduchou aplikaci prezentující základní možnosti Silverlightu. Jednalo se o seznam osob, který po výběru konkrétní osoby zobrazí její detaily.

Kromě základní práce s ovládacími prvky a samotným jazykem XAML, jsem poprvé přišel do styku s funkcí DataBindingu. Díky němu je možné celý seznam osob v XAML části programu nadefinovat jako:

```
<ListBox x:Name="lbOsoby" ItemsSource="{Binding SeznamOsob, Mode=OneWay}" />
```

Výpis 1: Základní definice ListBoxu

V kódu v pozadí stačí už jen přiřadit ListBoxu instanci třídy, která obsahuje Property s názvem SeznamOsob. Pokud tedy bude kolekce přímo ve třídě, která je kódem v pozadí pro ListBox, pak v konstruktoru stačí zavolat:

```
public partial class KartaOsob
{
    public List<Osoba> SeznamOsob
    {get;set;}

    public KartaOsob() //konstruktor
    {
        NapInSeznam();          // pomocná metoda
        lbOsoby.DataContext = this; // v našem případě je instancí,
                                    // ve které je vlastnost SeznamOsob,
                                    // třída sama
    }
}
```

Výpis 2: Přiřazení instance pro Binding ListBoxu

Nyní se zobrazí prvky kolekce jako položky ListBoxu. Samozřejmě implicitně ListBox zobrazí pro každou položku jeden řádek textu a poměrně logicky si vybere metodu ToString() položky. Stačí tuto metodu přetížit a určit, co chceme zobrazit.

Potřeboval jsem ale více flexibilní přístup. Co se týče obsahu je přetížení metody ToString() dostatečné, co do zobrazení ovšem nikoliv. Například je vhodné zvýraznit příjmení osoby a email naopak vytisknout malým písmem.

K tomuto požadavku jsem využil jednak stylování, které SL nabízí, a jednak další součást DataBindingu v SL nazývanou DataTemplates.

DataTemplates umožňuje v XAML kódu definovat u ListBoxu (samozřejmě i u jiných prvků) šablonu pro zobrazovanou položku.

```
<ListBox ItemsSource="{Binding SeznamOsob, Mode=OneWay}">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal" >
        <StackPanel Orientation="Vertical" >
          <TextBlock Text="(" /><TextBlock Text="{Binding Id}" /><TextBlock Text=")" />
          <TextBlock Text="{Binding Jmeno}" />
          <TextBlock Text="{Binding Prijmeni}" />
        </StackPanel>
        <TextBlock Text="{Binding Email}" />
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

Výpis 3: Použití DataTemplate

Každá položka bude nyní vykreslena pomocí této šablony. Předpokladem je, že položky v kolekci SeznamOsob obsahují veřejné Properties s názvy odpovídajícími těm uvedeným.

Zde se dostávám k dalším možnostem DataBindingu. Potřeboval jsem, aby změny, které budou provedeny u dané osoby, byly automaticky promítnuty i v ListBoxu. Řešení spočívá pouze v deklarativním zpřístupnění této funkce. Ta se nazývá DataBinding Mode a má tři stavy, viz tabulku 1.

OneTime	Hodnota bude přenesena pouze jednou, z vlastnosti do UI prvku
OneWay	Aktualizace (property → prvek) se provede po každé změně
TwoWay	Hodnota bude měněna oběma směry.

Tabulka 1: Módy bindingu

Mód se nastavuje v XAML části kódu (jako ve výpisu 3).

Vnitřně toto automatické provázání využívá událostí PropertyChanged z rozhraní INotifyPropertyChanged. Třída položek v seznamu tedy toto rozhraní musí implementovat a jednotlivé Property událost rozhraní vyvolávat (v Setteru). Díky tomu je zajištěna aktualizace změn v jednotlivých položkách. Pro aktualizaci samotné kolekce položek (přidání nebo odebrání položky) musí tato kolekce implementovat zmiňované rozhraní INotifyPropertyChanged a k tomu ještě rozhraní INotifyCollectionChanged. K těmto účelům slouží předimplementovaná třída ObservableCollection<T>, která již tato rozhraní implementuje.

DataBinding a ObservableCollection

Třída ObservableCollection se ovšem nechová podle očekávání. V RC verzi Silverlightu 2 se objevil problém s tím, že se kolekce nesynchronizovala s navázaným ListBoxem. Bylo nutné obejít aktualizaci kolekce pomocí metody UpdateLayout(), kterou poskytuje ListBox. Zdroj dat nastavit na hodnotu **null**, zavolat metodu UpdateLayout(), poté opět ListBoxu přiřadit původní zdroj dat. Ten je načten celý znovu a projeví se změna.

Tato chyba byla v RTW verzi opravena. Nyní nelze binding přepnout na OneTime a aktualizace tak vypnout. Samozřejmě pokud aktualizace nejsou potřeba, můžeme použít místo ObservableCollection například obyčejný generický List. Ovšem pokud bychom chtěli nastavení módu bindingu měnit během vývoje např. kvůli ladění, budeme muset měnit typ kolekce z ObservableCollection na List a zpět.

DataBinding v SL nabízí obrovské možnosti. Z dalších, které jsem při úpravách příkladu využil, můžu zmínit např. Convertery sloužící pro změny vázaných hodnot před zobrazením. [3]

2.1.2 Silverlight a komunikace s webovými službami

Aplikace samozřejmě nemohla data načítat staticky z kódu. Dalším úkolem tedy bylo provázat Silverlight s webovou službou a dynamicky načítat data z databáze na serveru nebo XML souboru na serveru. Pro samotnou webovou službu jsem měl použít technologii WCF.

Webová služba byla poměrně jednoduchá. ServiceContract (rozhraní) definoval několik OperationContracts (metod) pro získání kolekce Osob různými způsoby. Třídu Osoba jsem přemístil ze SL aplikace na WS jako DataContract.

Nyní jsem přistoupil k navázání SL aplikace s WCF službou. Prvním problémem se ukázal být binding WCF služby. Silverlight podporuje pouze základní basicHttpBinding (WCF nastavuje defaultně binding na wsHttpBinding).

V SL aplikaci jsem využil možností automatického generátoru ve VS2008, který z WSDL běžící služby automaticky vygeneruje proxy třídu. Zde se nachází druhý problém. SL aplikace neumožňuje blokující volání WS a je nutné využívat pouze asynchronních volání. To činí kód větších projektů velmi nepřehledným. Minimalizovat kód můžeme alespoň tak, že pro reakci na událost dokončení asynchronního volání využijeme lambda výrazy.

Zpracování výjimek

Na další velké omezení a problém, který trápí i vývojáře SL, jsem narazil při zpracování výjimek WS. Jedná se o tzv. „NotFound error“. [4]

Silverlight volá WCF službu a čeká na odpověď/výsledek. Během zpracování požadavku klienta může na WCF službě dojít k chybě, přičemž bude vyvolána vyjímka. Pro tyto účely WCF využívá FaultContracts. Při odchycení vyjímky je vyhozena (throw) nová instance FaultContract třídy, která může být uživatelem definovaného typu. V HTTP odpovědi je pak návratový kód 400 a obsahem je instance FaultContract s informacemi

o chybě. (FaultContract je mapován na SOAP specifikaci chyby, jedná se tedy o korektní, sémanticky správné, řešení oznámení chyby.)

Aplikace pak může v kódu očekávat vyvolání výjimky a podle jejího typu (příp. dalších informací) se dál chovat. V případě asynchronních volání je tato výjimka přenášena v parametru třídy AsyncCompletedEventArgs.

To ovšem není případ Silverlightu. Při jakékoliv chybě na WCF službě dostane Silverlight jen výjimku CommunicationException se zprávou „The remote server retuned an error: NotFound“. Toto nestandartní chování je způsobeno chováním prohlížečů. Prohlížeč totiž standartně předá jakémukoliv pluginu (v našem případě Silverlight) odpověď serveru jen v případě, že je návratový kód 200 OK. V opačném případě, nehledě na návratový kód a obsah zprávy, prohlížeč předá pluginu jen zmíněnou chybovou zprávu o tom, že server vrátil chybu. Silverlight se tedy nikdy nedozví jakou chybu od webové služby měl obdržet.

Vyřešit tento problém se dá několika způsoby. Žádný z nich není ideální a v principu jde vždy o předání chybového hlášení s návratovým kódem 200 OK. Výjimku je tedy nutné na službě odchytit, zpracovat a průběh metody ukončit korektně. [5]

- Nejprimitivnějším a dočasným způsobem může být vrácení objektu **null** namísto hodnoty.
- Podstatně lepším řešením je odeslat chybové hlášení jako jednu z proměnných nějakého vlastního obalovacího typu. To je ovšem míchání dat aplikace s daty chybovými.
- Další řešení spočívá ve využití výstupního parametru (out parameter) vlastního typu určeného pouze pro chyby. Zde zase mohou nastat potíže s kompatibilitou pokud by neměl být Silverlight jediným klientem naší WCF služby.

2.1.3 Validace

Při komunikaci s WS měla aplikace možnost načíst ze serveru data a ty pak upravovat. S úpravami hodnot bylo nutné validovat správnost vstupních dat.

V RTW verzi Silverlightu 2 jsou možnosti validace spjaté s nastavením Bindingu. Toto řešení je poněkud hůře použitelné. A bylo poměrně rychle nahrazeno lepšími způsoby.

Pro validaci tedy vyšel na vývojářském serveru Codeplex.com speciální toolkit s názvem Silverlight Validator & Input Toolkit [6]. Jedná se o knihovny, které rozšiřují základní funkčnost Silverlightu. Konkrétně jde o sedm nových prvků, pomocí kterých můžeme kontrolovat vstupní hodnoty formulářových polí před odesláním či potvrzením.

Použití je nesmírně jednoduché (především oproti původnímu řešení). Mezi tagy textového pole, které chceme validovat, stačí přidat jeden (nebo více) z nových prvků a určit podmínky pomocí atributů. Tedy např:

```
<toolkit:ValidatorManager x:Name="Group1" />

<TextBox x:Name="tbJmeno" FontSize="20">
   < toolkit:ValidatorService . Validator>
```

```
        <toolkit:RequiredValidator ManagerName="Group1" />
    </ toolkit:ValidatorService . Validator>
</TextBox>
```

Výpis 4: Přidání validátoru k TextBoxu

V kódu v pozadí pak stačí jen zavolat (call) metodu ValidateAll () na instanci ValidatorManager, která vrátí seznam chyb. Pokud seznam není prázdný, můžeme zareagovat:

```
List< Silverlight . Validators . ValidatorBase> validators = Group1.ValidateAll();
if ( validators .Count != null)
{
    MessageBox.Show("Nektera pole nejsou vyplnena spravne");
}
```

Výpis 5: Kontrola chyb u všech polí s validací

Výhoda tohoto řešení spočívá v udržení oddělení logiky a vzhledu a obrovské jednoduchosti použití. V původním řešení bylo potřeba měnit vzhled textových polí (nebo třeba přidávat tooltipy s nápovědou) až v kódu. V toolkitu každý validační prvek obsahuje pomocné atributy, pomocí kterých lze vše nastavit v markupu. Například atribut ErrorMessage nastaví tooltip při špatném vyplnění (a zase jej odstraní při správném vyplnění). Navíc je na Codeplexu možné stáhnout i zdrojové kódy, jež jsou připraveny k tvorbě vlastních validátorů.

Silverlight verze 3 byl v mnohém ještě vylepšen. Definuje nové ovládací prvky (např. DataFrom), které dynamicky samy vytváří formulářová pole pro navázanou entitu. Veškerá nastavení chování formuláře pro jednotlivá pole se děje pomocí atributů u vlastností dané entity. Nastavit pomocí atributů je možné mód DataBindingu, zobrazovaný text, popisek, validační požadavky a další. [7]

Příklad je uveden ve výpise 6.

```
public class Osoba
{
    [Display(Name = "Jméno :", Description = "Zadejte vaše jméno")]
    [Required(ErrorMessage = "Jméno musí být zadáno")]
    public string FirstName { get; set; }

    [Display(Name = "Příjmení :", Description = "Zadejte vaše příjmení")]
    public string LastName { get; set; }

    [Display(Name = "E-mail :", Description = "Zadejte váš email")]
    [RegularExpression(@"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*", ErrorMessage = "
        Špatný formát emailu")]
    public string Email { get; set; }

    [Display(Name = "Věk :", Description = "Zadejte váš věk")]
    [Range(0, 200, ErrorMessage = "Špatná hodnota věku. zadejte číslo mezi 0-200")]
    public int Age { get; set; }
}
```

Výpis 6: Entita připravená pro vazbu s prvkem DataFrom

2.2 Windows Workflow Foundation

Po SL měl můj zájem směřovat k technologii WF. Zde bylo úkolem vytvořit jednoduchou aplikaci prezentující všechny základní možnosti WF. [8]

Konkrétně se tedy jednalo o WPF hostující prostředí, umožňující načíst několik souborů s definovaným workflow pomocí XAML jazyka. Tyto načtené workflow pak spouštět, zastavovat, rušit, monitorovat jejich stav, uspat, při dalším spuštění aplikace obnovit uspaná workflow, sledovat průběh pomocí TrackingService a ukládat záznamy do DB a vyzkoušet komunikaci mezi hostujícím prostředím a workflow.

2.2.1 Hostování

Workflow, vytvořené pomocí WF, není samostatné a potřebuje k běhu hostující aplikaci. Pro hostování se pak nekladou žádné podmínky. Hostovat můžeme v konzolové, WinForm nebo WPF aplikaci. Hostující prostředí poskytuje například i Sharepoint. Workflow může řídit tok webové služby (asmx i WCF) nebo i webového serveru.

Ke spuštění je potřeba běhové prostředí – WorkflowRuntime. To umožní vytvořit instanci – WorkflowInstance. Tuto instanci lze následně ovládat a Runtime může reagovat na události spuštěné instance.

Moje aplikace (hostující prostředí pro WF) tedy byla vytvořena pomocí WPF. Načtení workflow ze souboru xoml umožňuje metoda CreateWorkflow() přijímající jako parametr XmlReader. Jednotlivé workflow potom stačilo udržovat v kolekci jako WorkflowRuntime.

2.2.2 Aktivity

Aktivita je základní stavební prvek WF. Může obsahovat další aktivity. Sekvence aktivit se nazývá workflow.

Součástí Visual Studia je sada již vytvořených aktivit. Mezi vytvořenými aktivitami jsou například IfElseActivity , WhileActivity pro řízení toku, ParallelActivity , SynchronizationScopeActivity pro paralelní zpracování, EventDrivenActivity, ListenActivity pro práci s událostmi nebo třeba RecieveActivity a SendActivity pro komunikaci s WCF.

Pro mé účely jsem vytvořil jednoduchá workflow, která měla dobu běhu několik sekund. K tomu jsem využil DelayActivity).

2.2.3 Služby

WF podporuje systém zásuvných modulů v podobě služeb. Služby, které jsou v daném WorkflowRuntime spuštěny, rozšiřují základní funkčnost WorkflowRuntime.

Využil jsem služby PersistentService, která umožňuje uložit aktuální stav workflow do databáze a později toto workflow obnovit.

Dále jsem použil službu TrackingService zaznamenávající události do databáze.

Problém nastal při použití obou těchto služeb naráz. Ukázalo se, že to není možné. Alespoň ne tak, jak by se dalo očekávat. Je nutné využít speciální službu zajišťující tuto funkci. Služba má opravdu orginální název: SharedConnectionWorkflowCommitWorkBatchService.

Poslední službou, se kterou aplikace umí pracovat, a kterou jsem ve své aplikaci využil, je služba ExternalDataExchangeService zprostředkovávající komunikaci mezi hostující aplikací a běžícím workflow.

2.2.4 Komunikace WF a hostující aplikace

Existuje několik způsobů jak komunikovat mezi hostující aplikací a běžícím workflow.

Host ⟶ workflow

Nejjednodušším způsobem je předání argumentů workflow při spuštění. Třída WorkflowRuntime nabízí metodu CreateWorkflow() se dvěma parametry, pro argumenty slouží druhý parametr. Argumenty předáváme v podobě generické kolekce Dictionary <**string**, **object**> . Klíčem je typ String a odpovídá názvu property ve workflow, které vytváříme. Hodnota je typu Object a odpovídá hodnotě předávané do property. Jedinou podmínkou je mít dané Property ve workflow, aby se mohly dané hodnoty správně předat.

Druhým způsobem jak z hostující aplikace něco workflow sdělit, je pomocí událostí. Workflow tedy musí mít definovány reakce na dané události a hostující aplikace pak může události vyvolávat.

Workflow ⟶ host

Opačným směrem lze samozřejmě také komunikovat. Slouží k tomu speciální aktivita – CallExternalMethod. V tomto případě je nutné využít zmíněnou službu ExternalDataExchangeService, vytvořit rozhraní, které bude hostující aplikace implementovat a workflow pomocí něj může na hostující aplikaci definovanou metodu vyvolat.

3 Závěr

3.1 Využité znalosti

Během průběhu praxe jsem se setkal výhradně s novými produkty a technologiemi společnosti Microsoft. Neustále jsem byl v kontaktu s programovacím jazykem C#. Nutné tedy byly především znalosti tohoto programovacího jazyka a .NET Frameworku. Základní seznámení s tímto jazykem a frameworkem jsem získal díky předmětu Programování v C# ve třetím semestru studia na VŠB-TUO.

Obecně jsem samozřejmě využil i veškeré zkušenosti s programováním, které jsem nabyl během studia na VŠB-TUO. Programování se týkalo celkem 18 předmětů během všech 6 semestrů, nepřímých znalostí tedy byla spousta.

3.2 Chybějící znalosti

Konkrétní znalosti o zkoumaných technologiích jsem ovšem neměl žádné. Jednalo se o poměrně nové technologie, které pro mě byly úplně neznámé a i úkoly byly postaveny tak, že bylo nutné zkoumat různé funkce a poznávat možnosti.

Windows Workflow Foundation je tu sice od konce roku 2006, ale o nových technologiích představených jako součást .NET frameworku 3.0 - WPF, WCF a WF - se začalo mluvit až s vydáním .NET Frameworku verze 3.5 na přelomu roku 2007 a 2008. Přesto se mluví především o WPF nebo WCF, WF zůstává nezmiňováno v pozadí.

Silverlight byl během mého studia na praxi stále ve vývoji a finální RTW verze přišla až 14. října 2008. Jeho obrovská propagace na internetu samotnými vývojáři z Microsoftu však proces učení maximálně zjednodušuje a dělá technologii velmi populární. A podle mého názoru zaslouženě.

3.3 Dosažené výsledky a zhodnocení

Časově byla praxe velmi náročná a skloubit praxi se školou bylo obtížné. Nicméně se mi díky praxi podařilo proniknout k novým technologiím a postupům, k nimž bych se jinak pravděpodobně vůbec nedostal.

Bohužel jsem neměl možnost si znalosti prakticky vyzkoušet v reálném použití. Příklady, které jsem programoval, byly čistě ukázkové k vyzkoušení možností, vlastností či funkčnosti.

4 Literatura

[1] Kvados. *Profil společnosti* [online]. 2006- [cit. 2009-01-05]. Dostupný z WWW: <http://www.kvados.cz/Profil/tabid/53/Default.aspx>.

[2] *Microsoft Silverlight* [online]. 2007- , last modified on 5 May 2009 [cit. 2009-04-20]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Silverlight>.

[3] LIBERTY, Jesse. *XAML,Data Binding - Jesse Liberty - Silverlight Geek* [online]. 2008- , May 04, 2009 [cit. 2009-04-28]. Dostupný z WWW: <http://silverlight.net/themes/blogs/marvin3/taglist.aspx?App=jesseliberty&Tags=XAML/Data+Binding>.

[4] HEUER, Tim. *Silverlight and Web Service Errors* [online]. 2008- , posted on October 01, 2008 [cit. 2009-04-28]. Dostupný z WWW: <http://timheuer.com/blog/archive/2008/10/01/silverlight-and-web-service-wcf-error-exceptions.aspx>.

[5] OSOVETSKY, Eugene. *Faults and Exceptions when using Web Services in Silverlight 2* [online]. 2008 , posted on September 2008 [cit. 2009-04-28]. Dostupný z WWW: <http://eugeneos.blogspot.com/2008/09/faults-and-exceptions-when-using-web.html>.

[6] SPO81RTY. *Silverlight Validator & Input Toolkit* [online]. 2008- , Last edited Nov 22 2008 [cit. 2009-04-28]. Dostupný z WWW: <http://silverlightvalidator.codeplex.com/>.

[7] PENDSE, Vikram. *Silverlight 3 : DataForm control features – Part 3* [online]. 2009- , posted on March 24, 2009 [cit. 2009-04-28]. Dostupný z WWW: <http://pendsevikram.blogspot.com/2009/03/silverlight-3-dataform-control-features.html>.

[8] *Windows Workflow Foundation* [online]. [2009] , last modified on 30 April 2009 [cit. 2009-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Windows_Workflow_Foundation>.